EXHIBIT 1

UNITED STATES DISTRICT COURT
NORTHERN DISTRICT OF CALIFORNIA
SAN FRANCISCO DIVISION

ORACLE AMERICA, INC.

Plaintiff,

v.

GOOGLE, INC.

Defendant.

Case No. 3:10-cv-03561-WHA

OPENING EXPERT REPORT OF RODERIC G. CATTELL, PH.D.

TABLE OF CONTENTS

I. INTRODUCTION

A. Case Background

1. I have been retained as an independent expert on behalf of Google in this litigation. I have agreed to a rate of $300 per hour for my time in connection with this case. My compensation does not depend in any way on the outcome of this litigation.

2. Google has engaged me to provide expert testimony about Application Programming Interfaces (APIs), including what APIs are, why APIs and the re-implementation of APIs are important, and industry practices regarding APIs.

3. At this time, I have not created any exhibits to be used as a summary of, or as support for, my opinions. I reserve the right to create any additional summaries, tutorials, demonstrations, charts, drawings, tables, and/or animations that may be appropriate to supplement and demonstrate my opinions if I am asked to testify at trial.

B. Professional Qualifications

4. In 1974, I graduated in the top 1% of my class at the University of Illinois, and was awarded a Bachelor of Science degree in Computer Science. I was also invited to join the Tau Beta Pi and Phi Beta Kappa honorary societies.

5. In 1978, I completed my dissertation, *Formalization and Automatic Derivation of Code Generators,* and was awarded a Ph.D. in Computer Science from Carnegie Mellon University. My dissertation won the ACM Outstanding Computer Science Doctoral Dissertation Award. This award is presented annually to the author or authors of the best doctoral dissertations in computer science.

6. In 1978, I joined Xerox PARC as a research staff member. I contributed to the Cedar programming environment and to the Cypress database system. The Cypress database system was one of the first client-server database systems, and one of my contributions to the

system was the interface—what today we would call an API—that client applications used to access the database server's functional capabilities.

7. At Xerox PARC, I was the primary architect and co-developer of an electronic mail database, a database editor/browser, and a spatial data manager, each of which used a Cypress database and made use of the Cypress database API that I had developed.

8. I also further developed the idea of using an entity-relationship model as an extension of the relational model for databases. This included defining an API on top of Cypress's lower-level relational storage model to allow users to make use of the new entity-relationship model. This provided a powerful basis for hypertext-like browsing of an underlying database, for spatial display of relationships, for graphical schema design, and for queries; these ideas were used in many successful database user interfaces in the industry.

9. After Xerox PARC, I was employed by Sun Microsystems between 1984 and 2007, initially as an Engineering Manager, later as a Second-Level Manager, and finally as a Distinguished Engineer.

10. As a Database Engineering Manager, in just 1.5 years and with the help of two engineering reports, I built and released the Sun Simplify product, an innovative graphical database front-end that provided entity-relationship database browsing, editing and schema design. Among other things, Sun Simplify included an API that I developed called the Entity Relationship Interface Convention, or ERIC for short.

11. I also coordinated the development, documentation, testing, marketing, and support of the Sun Simplify product, and was the initial manager for a larger follow-on project, SimplifySQL, which was a joint project with Ingres Corporation. This involved working with

Ingres Corporation to develop a new API that allowed the Sun technologies to communicate with database systems, including the Ingres database system.

12. In 1986, I was promoted to Second-Level Manager, and served as Sun's second-level manager for database back-end and front-end technology, building a group of 20 engineers and two first-line engineering managers. I managed engineering relationships with database management systems (DBMS) vendors and with major customers.

13. In 1987, I developed and successfully sold Sun management on a new relational database strategy for Sun, under which Sun would be "database agnostic"—*i.e.*, would work with many different database systems from different leading vendors. My team also worked to enhance the Sun OS API and implementation to improve performance of database systems. These efforts and other work by my team proved very effective, and Sun's database server revenue grew to a billion dollars in the subsequent 5 years.

14. In 1988, I was selected as a Sun Distinguished Engineer. At that time, there were only a handful of Distinguished Engineers at Sun, and I was the only Distinguished Engineer in the database group. My responsibilities included setting Sun's overall technical direction with respect to databases, and participating in various company projects involving database technology.

15. In 1989, Sun and several other companies cofounded the SQL Access Group (SAG). I was Sun's representative on SAG. The other companies that were founding members of SAG were Oracle, Informix, Ingres, DEC, Tandem and HP. SAG was created because relational database vendors had no API standard meeting the requirements of advanced database interfaces. SAG started development of the SQL Call Level Interface, which was later published as an X/Open specification. In 1992, Microsoft released version 1.0 of the ODBC specification, which

drew from portions of the X/Open SQL Call Level Interface specification. In 1997, Sun released the JDBC specification, which provides an API for connecting from Java to relational databases. JDBC drew from ODBC and the X/Open Call Level Interface specifications.

16. From 1991 through 1999, I was one of Sun's representatives to the Object Management Group (OMG). During this time, I contributed to the design of a number of OMG APIs, including the Relationship Service, the Persistence Service, the Query Service, the Naming Service, the Transaction Service, and other APIs. The OMG APIs are standards for a multi-vendor distributed computing environment allowing many components to be plugged together to build applications.

17. In 1991, I organized a meeting of representatives from the object database vendors Objectivity, Object Design, Versant, Ontos and Servio. At this meeting, we conceived the idea of what became the Object Database Management Group (ODMG), which included as members those object database vendors, Sun, and various other companies. I served as chair and as editor for the group. The primary goal of the ODMG was to define an API that allowed a developer to write portable applications for object database and object-relational mapping products. In order to do that, the data schema, programming language bindings, and data manipulation and query languages needed to be portable.

18. In 1995, I joined Sun's new JavaSoft Division, and started an "enterprise" Java group for server-side Java. I served as the initial architect for the J2EE platform, a collection of APIs and implementations that eventually evolved into what is now known as Java EE. I also co-created the JDBC API, contributed to EJB (Enterprise JavaBeans) and other J2EE APIs, and recruited the key architects and management that produced J2EE.

19. I also started and led the Java Blend object/relational mapping project in partnership with the Baan Company. The Java Blend product made use of the ODMG API and SQL to allow it to work with databases from many different vendors.

20. I also created and led an "expert group" process for JDBC, and subsequently helped define Sun's Java Community Process (JCP) based on that experience. The JCP is the means by which Sun (and now Oracle) has developed and continues to develop and revise standard Java technical specifications, reference implementations of those specifications and related test suites. Anyone can participate in the JCP, although deeper levels of participation require becoming a JCP member. JCP membership is not limited to Sun or Oracle employees, and indeed many JCP members have been and are neither Sun nor Oracle employees.

21. From 2003 to 2006, I was a deputy to Sun's Software CTO, evaluating projects and defining strategy, particularly with respect to Sun's internal and external data management requirements.

22. From 2006 to 2007, I was the chief architect for Sun's Database Technology Group, working with open source database systems and internal projects. Among other projects, I started the Java DB product, which was a reference implementation of the JDBC API.

23. Since leaving Sun in 2007, I have been an independent consultant in database systems, focusing on scalable distributed architecture, database product selection, competitive analysis, and patent litigation.

24. I am probably best known for my contributions in database and server software, including database scalability, enterprise Java, object/relational mapping, object-oriented databases, and database interfaces. I am the author of several dozen papers, five books

(including, with co-authors, *JDBC API Tutorial and Reference: Universal Data Access for the Java 2 Platform*), and am named as an inventor on eight U.S. patents.

25. I have authored or co-authored three publications in the last ten years:

i) *Challenges for Brain Emulation: Why is it so Difficult?,* with Alice Parker, Natural Intelligence 1 (3), International Neural Network Society, June 2012.

ii) *Ten Rules for Scalable Performance in Simple-Operation Datastores,* with Michael Stonebraker, Communications of the ACM 54 (6), June 2011.

iii) *Scalable SQL and NoSQL Data Stores,* ACM SIGMOD Record 39 (4), December 2010.

26. During the previous four years, I have not testified as an expert at trial or by deposition.

C. Materials Considered

27. This report is based on my professional experience and training, and my specialized knowledge of the matters set forth in this report. If called as a witness, I could and would testify competently to these matters.

II. SUMMARY OF OPINIONS

28. The Java APIs are the functional means used by Java language programmers to invoke commonly used functionality.

29. If Android required the use of different APIs to allow programmers to invoke the same functionality that is invoked by the Java APIs, that would be contrary to my expectations as a programmer writing in the Java programming language, and the expectations and demands of the programming community.

30. If I, as a programmer writing in the Java programming language, had to use different APIs to invoke the same functionality on different platforms, that would have many

negative consequences, and would be contrary to the expectations and demands of the programming community.

31. Re-implementing APIs is commonplace in the computing industry, and has long been consistent with custom and practice in the computing industry. Indeed, Oracle has itself reimplemented IBM's SQL APIs in its relational database products.

III. APIS AND THE VALUE OF THEIR REUSE

35. APIs allow software to communicate with other software. An API for a given piece of software reduces that software to its essential functionality, defining the operations the software is willing to perform for other software; the details of the inputs it will accept in performing those operations; the outputs it will generate as a result; and certain other specific details, such as ways the software might signal that an error has occurred.

36. To take an example, the hardware chips inside of a computer typically do not come preprogrammed with the ability to perform mathematical functions such as calculating a square root, or choosing the larger of two numbers. Those functions, however, are very commonly needed by computer programmers, and rather than requiring that a developer write a small program to perform those tasks every time they are needed, it makes more sense for someone to write them once, to compile that functionality into what a developer typically calls a "library," and to publish APIs for accessing that functionality.

37. Those APIs are, to computer scientists, simply part of the vocabulary that they can use when programming. For example, I know that if I am programming in the Java programming language, I can use the sqrt method, which is in the Math class of the java.lang package, to calculate the square root of a number. And I know that if I wanted to calculate the square root of a variable called x, I would include the text "Math.sqrt(x)" in my program. Similarly, when programming in the Java programming language, if I want to choose the larger

of the numbers represented by the variables a and b I would include the text “Math.max(a,b)” which would use the max method, which also is in the Math class of the java.lang package.

38. Because computers are very literal, I also know that my programming code would not work with these APIs if I were to type “squareroot(x)” instead of “sqrt(x)”. And if I typed “maximum(a,b)” instead of “max(a,b)” that wouldn’t work, either. Thus, when I am programming in the Java programming language, I always type “sqrt” if I want a square root, and “max” if I want the larger of two numbers. This is very different than if I were writing, say, a novel, where if I had to say almost the same thing several times, I would probably try to come up with a few different ways of saying it, just to keep the text from getting dull.

39. But if, hypothetically speaking, I had to type “sqrt” when I was programming in the Java language for the Oracle Java platform, and “squareroot” when I was programming in the Java language for the Android platform, that would be contrary to my expectations as a programmer writing in the Java programming language. Continuing on, if every company with a Java programming language product had to use different ways of naming the square root method—so that, for example, a company other than Oracle or Google might need to use “squarert”—the end result would be a bit of a disaster, for many reasons.

40. First, imagine I had written some original code for a project on the Oracle Java platform, and was later working on a second project, this time for the Android platform. To the extent I wanted to reuse portions of the code I had written for the first project for the second project, I might not be able to do so if the APIs for basic functionality (like calculating square roots or the larger of two numbers) are not the same. Instead, I would have to change various API uses to adapt the code from my first project for the second project. Not only would this take longer, but it would also introduce a significant risk of errors in my code, particularly if every

single function has a different name on every product that can be programmed using the Java programming language. This would be contrary to the expectations and demands of Java language programmers.

41. Second, imagine I am writing some new code. If I need to calculate a square root, I then am forced to pause and consider whether my new code is for the Oracle Java platform, the Android platform, or some other Java programming language context. This is a needless interruption in my stream of thought. And if there are many different options, with many different ways of calling the square root functionality, the interruption might require me to look up the right API for this particular context. This, too, would be contrary to the expectations and demands of Java language programmers.

42. Third, imagine I am a company creating a new product for which programs can be written using the Java programming language. If I am not allowed to use sqrt for my square root method (because that belongs to Oracle) or squareroot for my square root function (because that belongs to Google), or even squarert (because that belongs to some third company), it becomes very difficult for me to create another option. Imagine I'm the one hundredth company to enter this space—at that point, I might well be forced to come up with something unpalatable like "two-root" or "skwarerrut," simply because all the more obvious variants have already been claimed by others, and I don't want to risk a copyright claim by using someone else's less cumbersome variant. Such a product would, for the reasons already discussed, run contrary to the expectations and demands of Java language programmers.

43. The examples I have just given involve the hypothetical situation of being forced to use different names for a method in different contexts. But essentially the problems would arise if other differences were required. For example, if Android and other systems organized the

methods differently—perhaps putting all methods that start with “m” in a package called “android.m”—that would create similar problems. Or if Android methods required that the inputs be ordered differently, or used different errors (called “exceptions” in the Java programming language), that, too, would cause similar problems—and, again, would run contrary to the expectations and demands of Java language programmers.

44. In my opinion, this hypothetical world is bad for everyone. It’s bad for the developers, whose expectations about the methods available to them when they program in the Java language are violated, forcing them to remember different words and syntax for the same functions on different products for which programs can be written using the Java programming language—which increases the time it takes to develop programs, increases the risk of coding errors, and generally stands as an impediment to the creative work of developers. It’s bad for the new creators of products that can be programmed with the Java programming language, who are left with few if any commercially reasonable options for their core libraries—which decreases the incentive to create such new products, and for existing product creators to improve their products. And it’s bad for users, who likely will see fewer such products in the marketplace, and fewer applications for such products.

45. In my opinion, this hypothetical world is bad even for the first company to establish itself as a provider of an API. If others are forced to implement different APIs, developers can use those APIs with only the original provider’s product. That might well make the original product less attractive to developers, thus limiting its value. By way of contrast, if others are allowed to re-implement the first company’s APIs, developers are more willing to learn those APIs—especially if the products of the re-implementers are popular and thus provide access to many potential customers. In my opinion, Google’s adoption of the Java programming

language (including APIs from the Java core libraries) for the Android platform has greatly increased the developer community and market for Java. If Google were forced to change its approach to avoid an API copyright conflict, in my opinion that would be bad for everyone, including Oracle, whether Oracle realizes that or not.

46. Furthermore, allowing companies to create fiefdoms and control who can re-implement APIs is, in my opinion, bad for programming. To offer an analogy, allowing the company that first created an API to prevent others from re-implementing the API would be similar to allowing the first company that came up with a stick shift and a steering wheel to prevent other car companies from using the same interface to control their cars. Imagine if every time you wanted to buy a different make of car, you had to learn a new system for turning and shifting. Similarly, imagine if the first company to use the QWERTY layout for a typewriter could have prevented others from using the same layout—that would have made typewriters far less useful for everyone. As yet another example, if the first company that decided to have a “File” menu in its computer program with a “Print” option could have prevented later companies from using that interface for their programs, we would all constantly be trying to remember where the “Print” command is in every program we used, rather than being able to rely on the common interface that we have today. All of these examples highlight why APIs should be re-implementable, and how and why doing so is beneficial to consumers and the marketplace.

47. These are not simply my personal opinions. Based on my many years of experience in computer science, I believe these opinions are widely held by computer scientists. I know this from my decades of experience working closely with other computer scientists, including computer scientists at Sun. I also know this from having joined an amicus brief of many prominent computer scientists during the briefing of Google’s petition for a writ of

certiorari to the United States Supreme Court in this case[1] (that brief focuses on whether APIs are "copyrightable," but from the perspective of a computer scientist, the question is whether APIs can be freely re-implemented, not the legal doctrine that is the reason for that freedom). In addition, I know this based on the myriad examples of companies re-implementing APIs in the computing industry, such that it can safely be described as a common practice, and one that is consistent with long-standing industry custom.

48. For example, IBM's first personal computers included a BIOS—a basic input/output system. This was firmware that included computer code used to perform hardware initialization when a computer was turned on to provide runtime services for the operating system and computer programs. Many companies developed BIOSes that were compatible with the IBM BIOS without permission from IBM. This allowed the development of a thriving market for "IBM compatible" personal computers.

49. As another example, much of the modern Internet is powered by computers running the Linux operating system. The Linux operating system, and in fact multiple variants of the Linux operating system, re-implement hundreds of APIs from Unix. This allowed Linux to flourish, because developers were easily able to adapt code they had already written, and were able to write new programs using the knowledge they had from writing other Unix or Linux programs.

50. SQL is another good example, and one I know well from my experience in the database field. SQL stands for structured query language, and is an interface for accessing information in a relational database. SQL allows a programmer to write commands such as "SELECT AVG(SALARY) FROM EMPLOYEES", which computes the average employee

[1] The brief is publicly available at https://www.eff.org/document/amicus-brief-computer-scientists-scotus.

salary. SQL includes dozens of such commands that can be used by SQL programmers. IBM created SQL, but it has since been re-implemented by countless other database companies; in fact, Oracle was one of the first to do so. Re-implementing SQL allowed these other database companies to satisfy the expectations and demands of the database programming industry, and as a result, database programmers can leverage their knowledge of SQL to program databases for many systems, rather than having to learn an entirely new language for every database system. SQL is now implemented on database systems from IBM, Microsoft, MySQL, Oracle, PostgreSQL, Sybase, SAP, Teradata, and others.

51. The success of SQL as a common API for relational databases was one of the motivations for the ODMG's creation of a common API for object databases—the companies involved recognized that having a common API would be better for customers, and better for the object database market generally. Having a common API for object databases would meet the demands and expectations of object database programmers, just as SQL provided a common API for relational database programmers. The ODMG companies anticipated that if their object database products shared a common API, that would increase the market for their products, because developers would be able to use the common API to program many different object databases.

52. Programs known as "emulators" provide yet another example. For example, there is a program called WINE that allows users to run programs that were created for Microsoft Windows on a computer that is instead running the Linux or Mac operating system. To do this, the WINE developers had to re-implement the Windows APIs that Windows programs use to perform tasks such as creating windows, menus, and graphic displays. This means that consumers who have purchased programs for Windows computers can use them on Linux or

Mac computers, too. WINE is not the only emulator out there—there are many such emulators available for other purposes as well, for example to emulate video game APIs.

53. Indeed, it is commonplace for computer scientists to re-implement APIs that were created by others. For example, many programming languages that existed prior to the Java language have used "sqrt" as the API for calculating square roots, including FORTRAN, COBOL, ALGOL, Pascal, C and C++. That is almost certainly why my colleagues at Sun chose that method name for the square root API in the Java core libraries, even though a method name like getSquareRoot (starting with a verb, and using mixed case to indicate words) would have been more consistent with the naming conventions for Java methods. This is not an isolated example—any computer scientist who knows multiple computer languages can offer examples of commands that are very similar between languages, just as "sqrt" appears in many languages as the function or method name used to invoke the functionality of calculating a square root. For example, "sin" and "cos" were used in many programming languages that existed before the Java language for the mathematical sine and cosine trigonometric functions (e.g., FORTRAN, COBOL, ALGOL, Pascal, C and C++) , and "printf" is commonly used for printing formatted output (e.g., C and C++). This makes it easier for programmers to learn new programming languages.

IV. CONCLUSION

54. For all of these reasons, it is my opinion that allowing Google to re-implement Java APIs would be in the public interest, would satisfy industry expectations and demands, and would be consistent with longstanding custom and practice in the computing field.

Executed on the 8th of January, 2016 in Tiburon, California.

[illegible]

Roderic G. Cattell